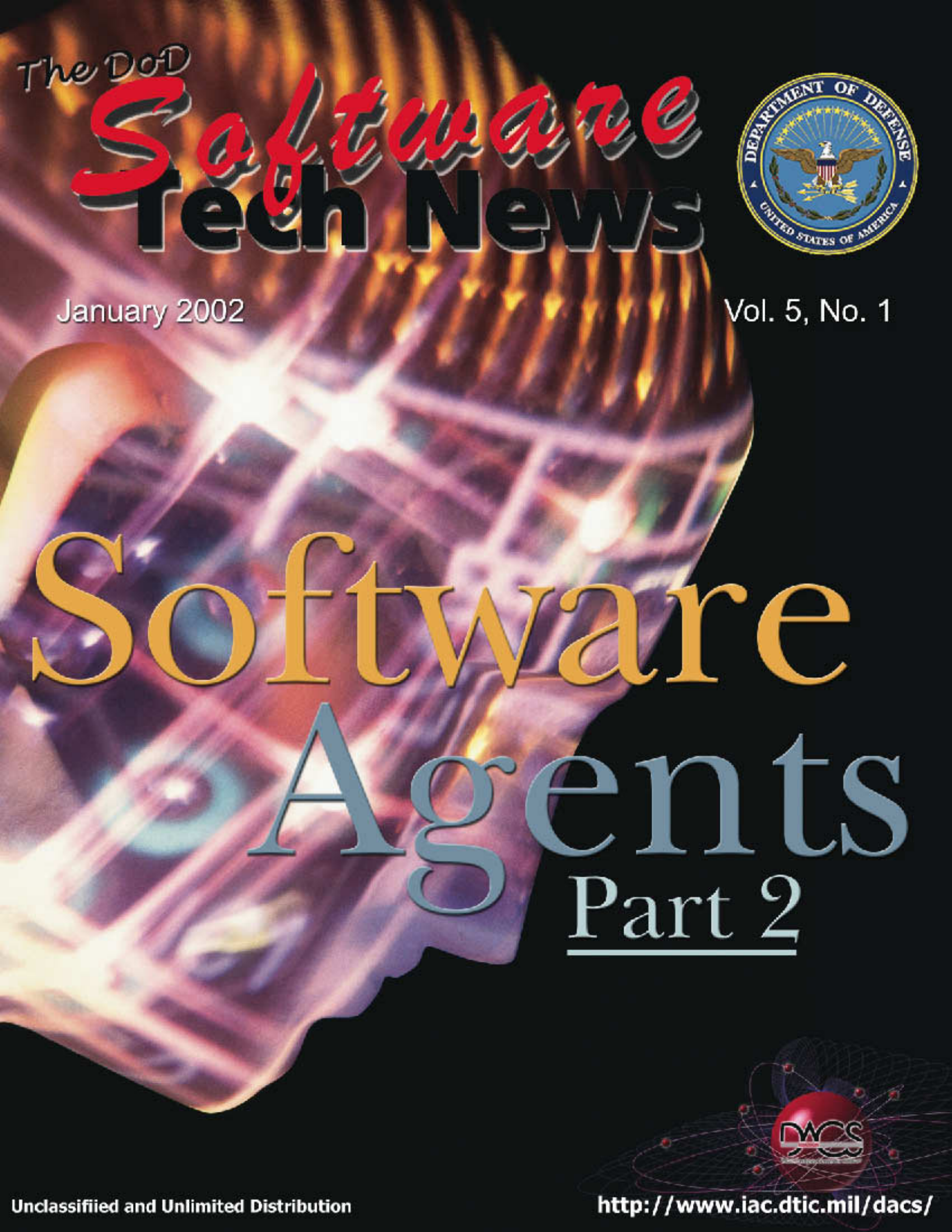
The DoD Software Tech News

January 2002

Vol. 5, No. 1

Software Agents

Part 2



http://www.iac.dtic.mil/dacs/

Strategies for Discovering Coordination Needs in Multi-Agent Systems

by Edmund H. Durfee, University of Michigan

Introduction

When multiple computational agents share a task environment, interactions between the agents generally arise. An agent might make a change to some feature of the environment that in turn impacts other agents, for example, or might commandeer a nonsharable resource that another agent desires. When the decisions that an agent makes might affect what other agents can or should decide to do, agents will typically be better off if they coordinate their decisions.

Numerous techniques exist for coordinating decisions about potential interactions. These include appealing to a higher authority agent in an organizational structure, instituting social laws that avoid dangerous interactions, using computational markets to converge on allocations, explicitly modeling teamwork concepts, using contracting protocols to strike bargains, and iteratively exchanging tentative plans until all constraints are satisfied. There is a rich literature on these and other mechanisms for coordinating agents; the interested reader can see [5].

However, each of these mechanisms takes as its starting point that the agents requiring coordination know, at the outset, either with whom they should coordinate, or what issues they should coordinate about. As examples, an organizational structure inherently defines how agents are related to each other, and a computational market corresponds to some resource or "good" that was somehow known to be contentious.

A central thrust of our research is in pushing back the boundaries of what is assumed known in a multiagent setting in order to bootstrap the coordination process. That is, we want to develop techniques by which agents can discover whom they should coordinate with, or what they should coordinate about, so that the rich variety of coordination techniques can then be employed. This article briefly summarizes some of our progress, results, and plans on this front.

Unintended Conflicts

An important case in which agents need to discover coordination needs is the following. Agents occupy an open, dynamic environment, and each agent has its own independent objectives. Yet, in pursuing its objective, an agent can unintentionally interfere with others, sometimes catastrophically. Therefore, it is important for each agent to discover whether something it is doing needs to be coordinated with others.

We have been studying coalition operations as an example application domain where this kind of problem arises. In a coalition, objectives and responsibilities are distributed among multiple functional teams, where operational choices by one team can infrequently and unintentionally affect another team. The repercussions of unintended interactions can range from merely delaying the accomplishment of objectives (such as waiting for assets that were unexpectedly borrowed by someone else) to more catastrophic outcomes (such as so-called friendly fire). We have been developing computational techniques in which

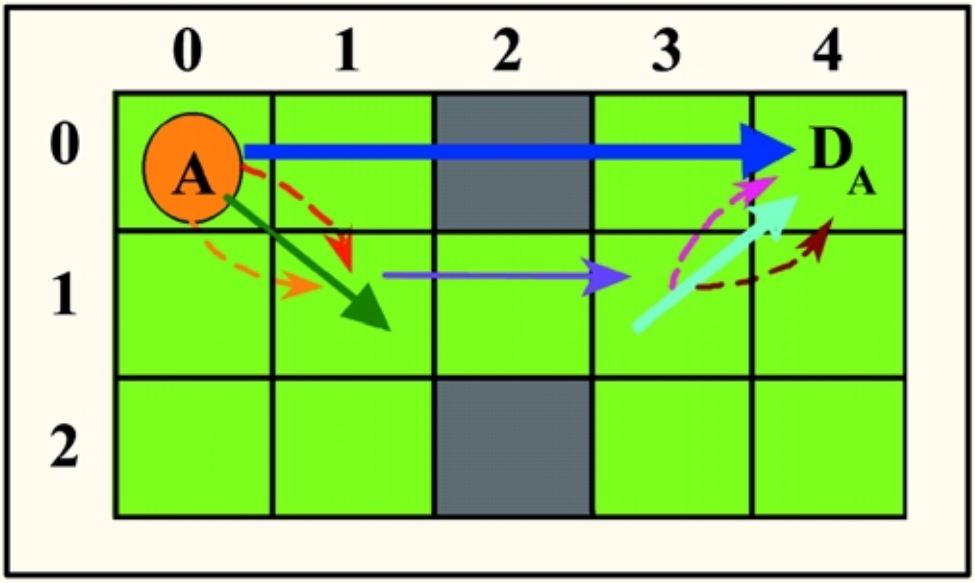


Figure 1: Example Movement Task

Continued on page 4

Coordination Needs in Multi-Agent Systems

Continued from page 3

each team is represented by a computational agent, and these agents predict the unintended interactions and resolve them before they occur. The resulting coordinated plans of the agents should be efficient (e.g., agents should not have to wait unnecessarily for others), flexible (e.g., agents should retain room in their plans to improvise around changing local circumstances), and realizable (e.g., agents should not have to message each other at runtime in a manner that outstrips communication capabilities).

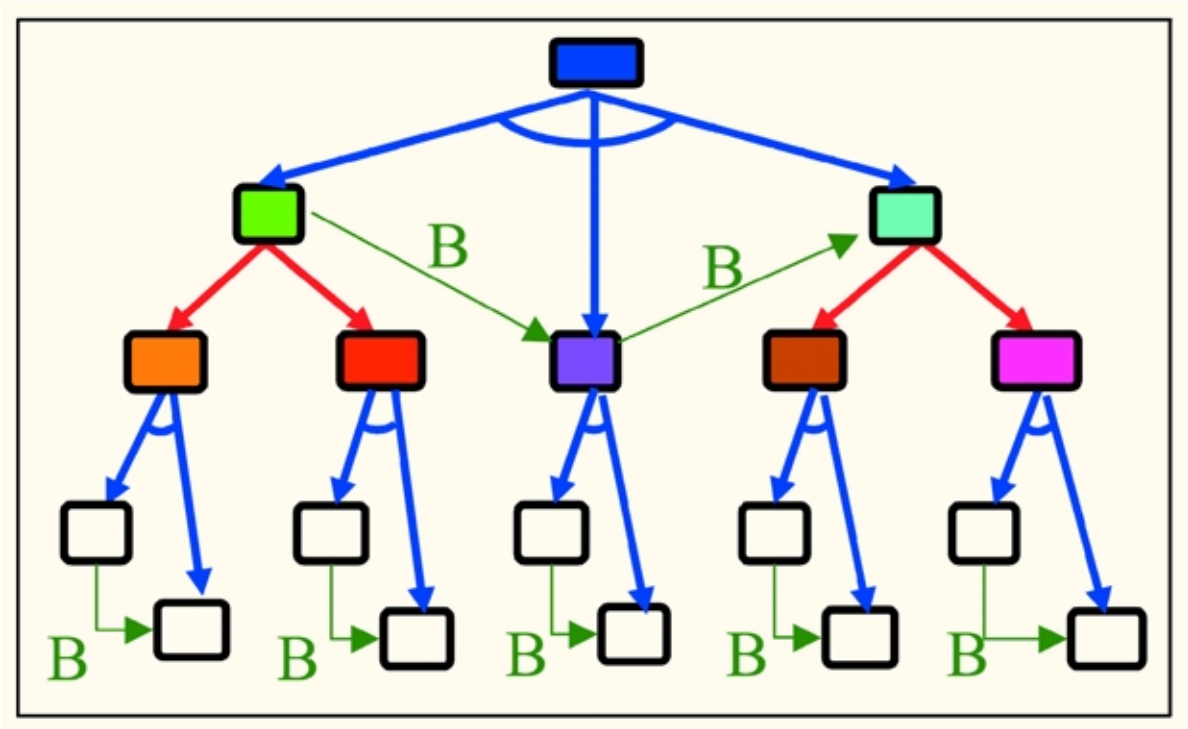


Figure 2: Example Hierarchical Task Network (HTN)

Conceptually, our techniques begin by assuming that each agent can represent its plans in a Hierarchical Task Network (HTN), capturing the possible decompositions of abstract plan steps into more detailed plans. As a simple example, consider the case of agent A moving through a grid world to reach a destination (Figure 1). The HTN for this agent is in Figure 2. At the most abstract level (blue arrow in Figure 1, blue node in the HTN), the plan is simply to go from the initial location to the destination. This is in turn composed of the three sequential steps of going to the door, through the door, and beyond the door (green, purple, and aqua arrows/nodes respectively). The ordering constraints are captured in the HTN (Figure 2) by the arrows labeled "B" for "before." For both the first and last step at this level, there are two ways of accomplishing the step. For example, for getting to the door, the red route or the orange route could be chosen. Each of these in turn can be decomposed into a sequence of two movements; red, for example, is to the right and then down.

An advantage of using the hierarchical representation is that each agent has, simultaneously, a model of itself at multiple levels of detail. In an open environment populated by numerous agents, being able to communicate about and exchange abstract information can enable agents to quickly determine which (small) subset of agents in the world they actually could potentially interact with (Figure 3a to Figure 3b). In the simple movement task example, for instance, the grid might be much larger, and the subset of agents is small whose planned movements, even abstractly defined, indicate a potential collision with agent A. For those agents, it might even be possible to impose constraints at the abstract level to ensure against unintended collisions, such as putting the overall plans in sequential order so that only one of the affected agents moves at a time. Or, for the remaining agents, additional details of the HTNs can be exchanged. As a result, agents that were potentially interacting might be determined to not interact at all, reducing the number of agents further and introducing constraints

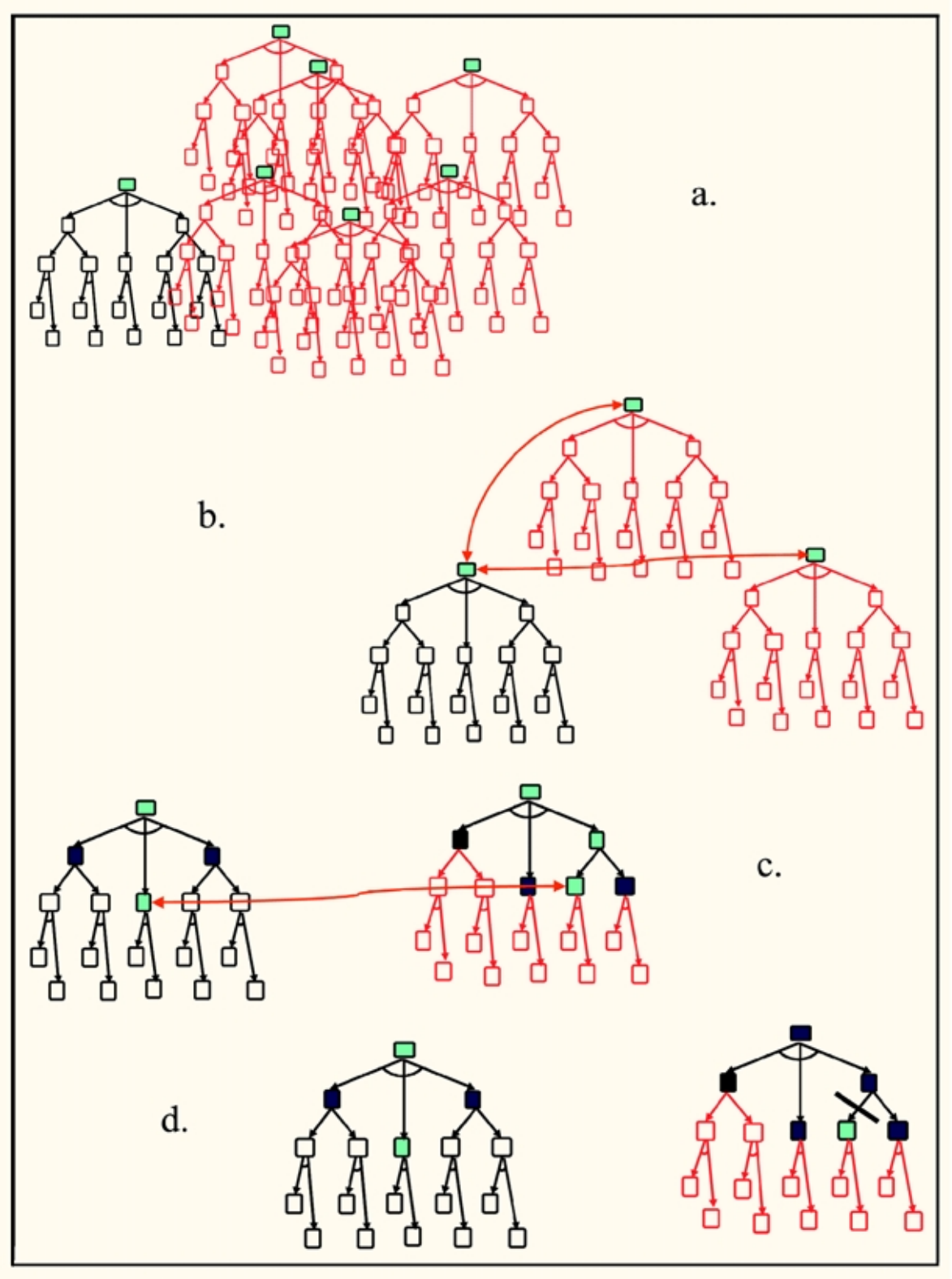


Figure 3: Top-Down Coordination Protocol Example

Continued on page 6

Coordination Needs in Multi-Agent Systems

Continued from page 5

between only substeps of plans leaving agents to do their other substeps as they wish (Figure 3c). Finally, further investigation might indicate that the potential conflict can be isolated to a particular choice that an agent might make; a commitment by the agent to forbid that choice leaves the plans coordinated without imposing any ordering constraints between the agents' plans at all (Figure 3d).

This example illustrates that, by working from the top down, agents can more efficiently identify and zoom in on the problematic interactions. By digging down deeply, they might be able to impose commitments (on relative timing or on choices of ways in which they will accomplish their tasks) that lead to very crisp coordination. However, in dynamic environments, sometimes it is better to impose constraints at more abstract levels: while this might require more sequential operation than desired, it also allows agents to avoid commitments to details that they might regret. As is intuitive in human coordination, each agent retains more flexibility for improvising when it makes more vague commitments to others. Moreover, digging down deeply requires more rounds of communication and analysis, so coordinating at abstract levels incurs less overhead. Among our ongoing research activities are developing methods for quantitatively evaluating tradeoffs between coordination "crispness", overhead, and flexibility.

We have developed techniques for formulating summaries in HTNs that permit the kind of top-down reasoning that we have just described, and have shown that such techniques can indeed much more efficiently coordinate agents [3]. These techniques have been shown to be sound and complete. At the cost of completeness, we have also developed a version of these techniques that can be used on-line [4]. The on-line techniques allow agents to postpone decisions about which of the alternative ways they will use to accomplish a task until that task is the next to be done. This in turn provides increased flexibility to the agents, leading to more reliable agent operation in dynamic domains than methods that require agents to make selections before execution begins.

Dealing with Centralization

The techniques just outlined have the feature that, to ensure that all possible interactions are detected and dealt with, some agent or agents need to compare all agents' most abstract plans. This implies that, at some point, information about all agents needs to be known in one place, which is antithetical to decentralized multiagent systems. Certainly, our current implementations rely on a central coordinator to discover potential agent interactions, although in principle once these are discovered the job of working with the agents to resolve the interactions can be delegated to multiple sub-agents, where each handles a different partition of the agent population.

How can we get around the need for centralization? Well, first of all, it should be noted that our use of centralization for detecting interactions does not imply that authority, or even knowledge about agent preferences, is centralized. In our model, the coordination process merely detects potential interactions and finds possible resolutions (more detailed resolutions as time goes on). To agree upon which resolution to use, the affected agents can employ any of the various coordination mechanisms mentioned at the beginning of this article. That is, these are appropriate once the agents know about the interactions and who is involved.

In turn, this suggests that one way of eliminating the centralization of the detection process requires that agents are initialized with some knowledge. For example, the organizational structure in which they reside might inherently partition the agents, such that coordination can be carried out in parallel in different partitions. Or agents might be initialized with knowledge of the possible actions of other agents that can be used to anticipate interactions. For example, our research is using these ideas to coordinate resource-limited agents in a multiagent world. In the simplest sense, a resource-limited agent needs to decide how to allocate its limited capabilities in order to meet its performance goals across the scope of worlds that it might encounter. By employing knowledge about what actions other agents might take in

particular situations, it can better predict what worlds it might encounter, and can even use its uncertainty to focus communication with those other agents to ask them which of the alternative actions they plan to take for a critical situation. Such communications could also permit agents to avoid taking redundant actions in situations where they would react the same way. In the long run, agents can even engage in negotiation to convince others to favorably change how they react to particular circumstances.

Congregating over Mutual Concerns

An alternative means of determining coordination needs, instead of centralizing information or inherently distributing key coordination knowledge, is to instead permit coordination needs to be discovered through interactions. While this would be inappropriate in applications where uncoordinated interactions could be catastrophic (such as when friendly fire arises in coalition operations), there are many applications where the consequences of poor coordination are not so dire.

Consider, for example, interactions among groups of people with similar interests, such as in an electronic newsgroup. A well-defined group permits an efficient exchange of relevant information among interested people, with a minimum of tangential communications that waste readers' time. A poorly-defined group, on the other hand, wastes the reader's time and might lose readership quickly, but with no significant lasting effects on the participants. In this case, then, it is possible that people might congregate around newsgroup topics in an emergent way, through experimentation and exploration in the space, until they converge on relatively stable newsgroups that lead to productive interactions.

We have been conducting research in understanding the dynamic processes of congregating in open environments [1]. In our model, agents move among congregations until they find places where they are satisfied, where satisfaction depends on the other members of their congregation. Since these other agents are also moving around to find satisfactory congregations, agents are engaged in non-stationary ("moving target") search. In general, convergence in such systems is slow if it happens at all, and we have been studying mechanisms that enhance convergence such as: varying the movement costs of agents so that some "hold still" while others move; allowing like-minded agents to move as a coalition; giving agents the ability to remember and return to previously-experienced congregations; and allowing agents in a congregation to summarize their common interests and advertise this information to other agents.

As a specific form of congregating, we have been particularly interested in information economies, where competing producers of information goods must bundle and price their goods so as to attract (a subset of) the information consumers. Where a producer ends up in the product-and-price space is influenced not only by the consumer preferences, but also by the positioning decisions of other producers. Among our research results are that we have defined some of the conditions that promote the discovery of niche markets in the information economy, such that producers engage in stable relationships with an interested subset of the consumer population, and avoid mutually-harmful interactions (price wars) with other producers [2]. As suggested above, the price paid for this decentralized technique for discovering which agents should coordinate (interact) with each other is that, on the way to the ultimate mutually profitable result, producers will sometimes compete with each other and do poorly temporarily as a result.

Summary and Future Directions

In this article, we have claimed that, while powerful techniques exist for coordinating agents that already know whom or about what to coordinate, there are still many issues that need to be explored in designing efficient mechanisms by which to determine what needs to be coordinated in the first place. We briefly described some mechanisms that we are exploring for this purpose. One of these involves agents iteratively exchanging plan information at increasingly detailed levels to isolate potential interactions

Continued on page 8

Coordination Needs in Multi-Agent Systems

Continued from page 7

and impose effective commitments to resolve conflicts. Another, on the other hand, permits suboptimal interactions to occur, and allows the agent population to self-organize, over time, into congregations that emphasize beneficial interactions.

There are many directions in which we are, or are considering, extending these research activities. We need to develop heuristic means by which agents can decide on the level of detail at which they should coordinate, and metrics for comparing alternative coordination decisions in uncertain environments. We need to extend the soundness and completeness proofs, as well as the complexity analyses, of the techniques as we continue to augment and improve them. Coordination commitments that are derived between agents should be generalized and remembered to form the core of a suite of team plans, and the processes by which coordination needs are discovered should apply not only between agents but also between agent teams. Finally, these techniques need to be implemented and evaluated in the context of challenging applications, such as in the domain of coordinating coalition operations.

Acknowledgements

The ideas and results described in this article were developed with numerous collaborators. In particular, I'd like to thank my students, including Brad Clement, Pradeep Pappachan, Chris Brooks, Haksun Li, and Jeff Cox. The work was supported, in part, by DARPA under the Control of Agent-Based Systems Initiative (F30602-98-20142), by DARPA under the Automated Negotiating Teams Initiative (subcontract to Honeywell on F30602-00-C-0017), and by NSF grant IIS-9872057.

About the Author

Dr. Edmund H. Durfee is currently a Professor of Electrical Engineering and Computer Science, and of Information, at the University of Michigan. He received a Ph.D. degree in Computer Science from the University of Massachusetts in 1987 and joined the EECS Department at the University of Michigan in 1988. His area of teaching and research is artificial intelligence, multi-agent systems, and real-time intelligent control. To date he has published over 100 journal and conference papers, and has served in various capacities such as the program chair (1998) and conference chair (2000) for the International Conference on MultiAgent Systems. He has received a Presidential Young Investigator Award from the NSF (1991), is a senior member of IEEE, and has been elected as a Fellow of the American Association of Artificial Intelligence (AAAI).

Contact Information

Dr. Edmund H. Durfee
EECS Department
University of Michigan
Ann Arbor, MI 48109

durfee@umich.edu

References

[1] Christopher H. Brooks, Edmund H. Durfee and Aaron Armstrong. "An Introduction to Congregating in MultiagentSystems", *Proceedings of the Fourth International Conference on MultiAgent Systems* (*ICMAS-2000*), pages 79-86, July 2000.

[2] Christopher H. Brooks, Edmund H. Durfee and Rajarshi Das. "Price Wars and Niche Discovery in an Information Economy", *Proceedings of the ACM Conference on Electronic Commerce 2000* , October 2000.

[3] Bradley J. Clement and Edmund H. Durfee. "Theory for Coordinating Concurrent Hierarchical Planning Agents Using Summary Information.", *Proceedings of the National Conference on Artificial Intelligence* (*AAAI-99*), pages 495-502, July 1999.

[4] Pradeep M. Pappachan and Edmund H. Durfee. "A satisficing multiagent plan coordination algorithm for dynamic domains", *Proceedings of the ACM Conference on Autonomous Agents (Agents-Ol)*, June 2001.

[5] Gerhard Weiss (editor). Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. MIT Press: Cambridge Massachusetts, 1999.

Semantic Interoperability Among Agents

by Drew McDermott, Yale University

Introduction

Although there are many definitions of "agent," one of the most interesting is "self describing program." If a web-based program provides a detailed description of flow to interact with it, then it is an agent, and other programs can use the information it provides to decide at run time how to interact with it. Extensions of existing *planning algorithms* can be used to construct sequences of actions, or more elaborate action structures, for accomplishing goals by sending requests to an agent. One obstacle to this happy scenario is that not all agents will describe themselves, or their ideal partners in the same notation, or, using the fashionable word, using the same *ontology.* In that case, the agents can't be coupled unless an ontology translation can be found. Taking a logical point of view, the problem can be thought of as managing a *merged ontology* containing *bridging axioms* that relate the terms of one ontology to the terms of the other, not necessarily one-to-one.

Although many people are excited about "agent technology," few people agree about what agent technology actually might be, or even what the word "agent" might mean. In this article, I mean something very specific: an agent is a computer program accompanied by a description of what it does and how to use it. The description is detailed and perspicuous enough that another program can interact with the agent without having to be explicitly programmed to do so. In other words, an agent is a *self-describing program.* For example, a Wabash.com[1] of the future might provide enough information that an automated shopping program could conduct a transaction with it by deducing from scratch how one asks Wabash.com for information, how one tells it what one wants, how one tells it the appropriate billing information, and so forth. The shopping program might itself be an agent, of course, and to avoid seeming to make irrelevant distinctions I will use the word "agent" as though all programs were self-describing unless proven otherwise.

The need for this kind of self description is evident to many people. For example, companies active on the Internet are interested in supporting automated business-to-business transactions by making their websites self-describing. This interest has led to the creation of the UDDI (Universal Description, Discovery, and Integration) consortium that is developing notations for Website self-description (see www.uddi.or). UDDI, and related notations such as WSDL and SOAP [2, 1] use XML [4] as their basic syntactic vehicle. They tend to focus on representing attributes of web agents such as where they are located, what protocols they use to communicate, and such.

In this article I will argue that one can aim higher. Rather than represent service descriptions with simple attribute-value tables, we should be able to say things like: "To put an item in the shopping cart, send a message of type Put-in-shopping-cart and parts Product-id and Quantity." One way to formalize this statement is with a notation like this:

```
(:action (put-in-shopping-cart id - Product-id quantity - Integer)
        :effect (quart-in-cart id quantity))
```

which is derived from PDDL (Planning Domain Definition Language) [6], a notation for defining all the legal actions in a domain and what their effects are.

The use of such notations does not mean we are giving up on XML, which has earned wide popularity as a message-exchange medium on the Internet. But XML is defined to be "machine-readable and human-tolerable." That is, it isn't completely opaque to people, but it isn't supposed to be easy for them to produce. One way to encode the definition above in XML is this:

Continued on page 10

[1] A little known competitor to a large on-line bookseller named after a river.

Semantic Interoperability Among Agents

Continued from page 9

```
<Action>
    <name resource=" ecom: put-in-shopping-cart " />
    <params>
        <rdf: Seq>
            <rdf: li>
                <VarID="v1" name="id">
                    <type resource="ecom:Product-id"/>
                </Var>
            </rdf :li>
            <rdf: li>
                <VarID= " v2 " name= "quant " >
                    <type resource="ecom:Integer"/>
                </Var>
            </rdf: li>
        </rdf: Seq>
    </params>
    <effect>
        <Predication>
            <Subj resource="#v1"/>
            <Pred re sourc e= " e com: quant - in- cart " />
            <Obj resource="#v2"/>
        </Predication>
    </effect>
</Action>
```

This is actually in RDF [5], a set of conventions for using XML syntax to describe arbitrary objects. Although it is possible to understand it, and even type it, there is no real reason for people to ever touch RDF or XML. Computers like these notations because they unambiguously specify the syntactic hierarchy of an expression, even for programs that know little about the content. People tend to prefer conciseness and layout in order to grasp the structure and meaning of an expression. For the rest of this paper, I'll use logic-based notations with Lisp syntax, but rest assured that when it comes time for agents to exchange information, they will probably do it with notations containing a lot more angle brackets.

There are several different phases involved in having two agents hook up. The first is the *advertising/ search* phase. It is based on the assumption that agents with something to sell will post descriptions of their abilities in central registries, where agents trying to solve a problem can find them. This is also sometimes called the *brokering* phase, based on the plausible assumption that the registry plays an active role in coupling the two agents together.

Our focus is on what happens after the brokering phase. One can generally assume that the agent descriptions used for advertising and search are fairly "shallow." The descriptions of what agent 1 wants and what agent 2 has are in some broad vocabulary that enables one to distinguish book sellers from bookies. Once the decision has been made to couple two agents together, a more detailed description comes into play. It's at this point that action definitions like that for **put-in-shopping-cart are made** available. If a merchant has provided definitions for all the possible actions a customer can take, then the customer is faced with a problem of the following form:

> *Given the current situation, and definitions of the possible actions in every situation, find a sequence of actions that will achieve a given goal.*

This is in essence what is called in AI a *planning problem*. I will use the phrase *planning phase* for the process of solving this problem, that is, finding an action sequence. After the planning phase comes the *execution phase,* when the sequence of actions is actually carried out. It is reasonable (we hope) to assume that the planning agent will succeed if it

executes the plan; but there may well be situations where the plan exits prematurely with some sort of failure indication. In that case the agent may give up, or *replan,* starting from the situation it finds itself in halfway through the original plan.

As an example of a planning problem, suppose the buying agent wants to own a copy of *Ubik* by Philip K. Dick, but spend less than twelve dollars. Through a broker, it finds a merchant, Wabash.com, that seems to be in the right business. It asks Wabash.com for its description, computes for a while, and then generates the following plan:

This plan is much simplified, but exhibits certain key features:

1. Steps can be tagged with arbitrary symbolic names, which enables the agent to refer to the step-value of previous steps.
2. The step-value of a receive step is the message received. The step-value of a send step is a little less intuitive: it's a "message id" that allows the sending agent to match up replies with sends. That is, in (receive u' i), the argument i should be the message id of the send the received message is in response to.
3. The plan is not guaranteed to succeed. The verify steps check to see if assumptions are fulfilled; if not, the plan fails. The planning agent will have to replan, or perhaps find another merchant agent to deal with.

```
(series (tag s1 (send Wabash. con
      (query-in-stock ( (author "Philip K. Dick")
                        (title "Ubik")) ) ) )
(tag s2 (receive Wabash.com (step-value s1)))
(verify (exists (pid - Product-id)
      (= (step-value s2)
      (in-stock-reply yes pid) ) ) )
(send Wabash. con
      (put-in-shopping-cart (! pid (step-value s2)
                        1)))
(tag s4 (send Wabash. cam
      (payment-method (credit-card "9876 6802 2963 3715")))
(tag s5 (receive Wabash. cam (step-value s4) ) )
(verify (= (step-value s5) (payment-method-status authorized)))
(tag s6 (send Wabash.com (confirm-purchase)))
(tag s7 (receive Wabash.com (step-value s6)))
(verify (= (step-value s7) (purchase-status confirmed)))
```

A more realistic planner could produce a branching plan, with alternative continuations after a run-time test. For instance, the plan might say to try the usual credit card, and if it is refused, try sending a backup credit-card number.

This is a rich area for research, but I want to focus on a different set of issues, concerned with how agents cope with differences in vocabulary and terms constructed from it. There is no guarantee that when two agents encounter each other they will talk about the same thing using exactly the same terms. It's not that easy to create notations for agent descriptions, and the more diverse the population of agents to be described gets, the harder it gets to find a notation that everyone involved can agree on. If self-describing agents become a reality at all, it is likely that notations will be centered around particular industries or other types of institutions (military, educational, and such). Within a community that shares a notation, communication will be fairly straightforward. Between communities, it will be considerably more difficult.

The word "ontology" is in fashion for talking about vocabularies and notations. Philosophers use the word, as a singular noun only, to mean the philosophy of being; in the representation business, we often contemplate multiple competing "ontologies." That's because the word has come to mean "How objects in a domain are named,

Continued on page 12

Semantic Interoperability Among Agents

Continued from page 11

classified, and dissected." The link to the philosophy of being is the idea that "to be is to have a name," so that what you don't give a name to might as well not exist. For example, in the bookselling business you distinguish between the paperback and hardcover versions of a book, but you don't distinguish between the 7th and 8th printing of the hardcover edition. If someone wants to program their agent to buy a book from the 7th printing, he is out of luck. Wabash.com's agent will never know what he is talking about. Printings may as well not exist as far as it is concerned.

There are other cases where agents from overlapping domains do talk about the same things, but in different ways. Here they do have a chance to communicate, provided a way can be found to translate between ontologies. This is a very difficult problem, much harder, for instance, than the planning problem we sketched above, which is hard enough. The reason it is so difficult is that it often requires subtle judgments about the relationships between the meanings of formulas in one notation and the meanings of formulas in another. Furthermore, there is no obvious "oracle" that will make these judgments. We cannot assume that there is an overarching (possibly "global") ontology that serves as a court of appeals for semantic judgments. There are times when such a strategy will work, but only after someone has provided a translation from each of the disparate ontologies to the overarching framework, and there is no reason to expect either of these translation tasks to be any easier than the one we started with. Indeed, the more the overarching framework encompasses, the harder it will be to relate local ontologies to it. I fence the work of ontology reconciliation inevitably involves a human being to do the heavy lifting. The most we can hope for is to provide a formal definition of the problem, and software tools[2] to aid in solving it.

The problem of ontology translation is complicated by the fact that different ontologies are expressed in radically different notations, from relational databases to semantic networks. This diversity makes it seem as if a key part of the problem is expressing mappings between arbitrary data structures. Our research group is going in a different direction. We don't assume that "mapping" something to something else is the crux of the matter, but instead that the problem is to infer content expressible in one ontology from content expressed in the other.

We start with the postulate that the different syntactic forms used by different ontologies can be factored out, allowing the problem to be phrased at the content level only. What this assumption comes down to is the idea that everything expressed in an ontology can be expressed in a neutral logical syntax, so that the only way it can differ from other ontologies is in its vocabulary. For the rest of this paper, I will assume that all facts are expressed in terms of *formal theories,* each of which contains the following elements:

1. A set of *types.*
2. A set of *symbols,* each with a type.
3. A set of *axioms* involving the symbols.

Once we have cleared away the syntactic underbrush, the ontology-transformation problem becomes much clearer. Suppose one bookseller has a theory O_1 with a predicate (`in-stock x - Book t - Duration`), meaning that x is in stock and may be shipped in time t. Another bookseller expresses the same information in its theory O_2, with two predicates, (`in-stock y - Book`) and (`deliverable d - Duration y - Book`). We are presented with a dataset D_1 that is in terms of O_1, which contains fragments such as

```
(:constants Ubik Ulysses - Book)
(:axioms (in-stock Ubik (* 4 day))
         (in-stock Ulysses (* 24 hour))
         ...)
```

To translate this into an equivalent dataset that uses O_2, we must at least find a translation for the axioms. The types and constants need to be handled as well, but we'll ignore that.

[2] Such as those described by [9].

With this narrow focus, it becomes almost obvious how to proceed: Treat the problem as a deduction from the terms of one theory to the terms of the other. That is, combine the two theories by "brute force," tagging every symbol with a subscript indicating which theory it comes from. Then all we need to do is supply a "bridging axiom" such as

```
(forall (b t) (iff (in-stock1 b t)
(and (in-stock2 b)
(deliverables t b))))
```

which we can use to translate every axiom in D_1, or any other dataset. More precisely, we can use it to augment the contents of D_2. Any time we need an instance of (in-stock$_2$ x) and (deliverables y x), the bridging axiom will tell us that (in-stock$_2$ Ubik) and (deliverable$_2$ (* 24 hr) Ubik) are true (and maybe other propositions as well). (the *lifting axioms* of [3].)

It seems as if we have lost sight of our original goal. We were looking for an approach to *transforming* ontologies, and we seem to have found a way of *merging* ontologies. Actually, that is not such a bad place to be. It suggests that the case of two ontologies is not special; we might well want to merge three or more. There are efficiency issues, but they are not that different from those that arise in importing modules from programming-language libraries.

Still, one is likely to feel that problems like the one just given are much too easy. In realistic cases, two ontologies will "carve the world up differently." They may have different "granularity," meaning that one makes finer distinctions than the other; of course, 0_1 might make finer distinctions than 0_2 in one respect, coarser distinctions in another. Here's an example: suppose 0_1 is the ontology we have been drawing examples from, a standard for the mainstream book industry. Now suppose 0_2 is an ontology used by the *rare* book industry. The main difference is that the rare-book people deal in individual books, each with its own provenance and special features (e.g., an autograph by the author). Hence the word "book" means different things to these two groups. For the mainstream group, a book is an abstract object, of which there are assumed to be many copies. If a customer buys a book, it is assumed that he or she doesn't care which copy is sent, provided it's in good condition. For the rare-book industry, a book is a particular object. It may be an "instance" of an abstract book, but this is not a defining fact about it.

For example, if you buy Walt Whitman's Leaves of Grass from Wabash.com, you can probably choose from different publishers, different durabilities (hardcover vs. paperback, page weight), different prices, and various other features (scholarly annotations, large print, spiral binding, etc.). However, you certainly can't choose exactly which copy you will receive of the book you ordered; and you probably can't choose which poems are included, even though Whitman revised the book throughout his life. The versions in print today include the last version of each poem included in any edition.

If you buy the book from RareBooks.com, then there is no such thing as an abstract book of which you wish to purchase a copy. Instead, every concrete instance of Leaves of Grass must be judged on its own merits. Indeed, making this purchase is hardly a job for an automated agent, although it could be useful to set up an agent to tell you when a possibly interesting copy comes into the shop.

Let's look at all this more formally. Suppose that the planning agent uses the industry-standard ontology (0_1), and the broker puts it in touch with RareBooks.com, with a note that although it bills itself as selling books, its service description uses a different ontology (0_2). If after trying more accessible sources the planning agent's goal can't be achieved, then the broker may search for an existing ontology transformation, or merge, that can be used to translate RareBooks's service description from 0_2 to 0_1.[3]

Continued on page 14

[3]If it can't find one, all it can do is notify the maintainers of the ontologies of the problem; there is no way for the broker, the planning agent, or the end user to find a transformation on the fly.

Semantic Interoperability Among Agents

Continued from page 13

Let us sketch what some of the bridging axioms between O_1 and O_2 might look like. In particular, we need to infer instances of (is $Book_1$ *x*) given various objects of type $Book_2$ with various properties. Objects of type $Book_1$ we will call *commodity books;* an example is the Pocket Books edition of Mein Kampf. Objects of type $Book_2$ we will call *collectable books;* an example is a copy of Mein Kampf once owned by Josef Stalin. It is roughly true that many, but not all, rare books can be thought of as instances of particular commodity books. Two rare books are instances of the same commodity book if they have the same publisher, the same title, the "same" contents, and the same characteristics (e.g., hardcover, large print, and such).[4] We can produce the following bridge axioms:

```
(: functions (book-type x - Book2) - Book1)
 (:axioms (forall (b12 b22 - Book2)
      (iff (and   (= (publishers b12)
                     (publishers b22))
                  (= (titles b12) (titles b22))
                  (= (phys-charac2 b12)
                     (phys-charac2 b22))
                  (< (revision-dif2 b12 b22) 1~5))
      (= (book-type b12)   (book-type b22))))
(forall (b2 - Book2)
      (=   (buy2 b2)
      (buy1 (book-type b2))))
```

This should all be self-explanatory, except for the predicate revision-dif, which we suppose is in use in the rare book business to express how many revisions are found between an earlier and later copy of an author's work. We have introduced a new function book-type, which maps individual collectable books to their types, which are commodity books.

For axioms such as these to do the planning agent any good, it must be possible for the planning agent to use them to translate a rare-book dealer's service description. Suppose the agent is trying to buy a copy of Lady Chatterly's Other Lover, a little known sequel to D.H. Lawrence's famous work[5]. Having exhausted the usual sources, it attempts to deal with RareBooks.com. It must find a plan in the merged ontology, then execute it.

Here are the main points I have tried to make:

1. Interagent communication requires a sophisticated level of representation of knowledge states, action definitions, and plans.
2. This representation can only be logic-based; no other notation has the expressive power. Embedding this logic in some form of XML/RDF/DAML notation is a good idea for web-based agents, but puts nontrivial demands on the representational power of those notations.
3. In spite of the expressivity, there are algorithms for manipulating logic-based expressions that might overcome computational complexity problems.
4. In particular, planning algorithms are a natural fit to the idea of a *service description.* The service description specifies the possible interactions with an agent; a plan is a sequence of interactions to achieve a specific goal. Finding such plans is more or less what planning algorithms do.
5. One of the worst obstacles to allowing agents to use each others' descriptions to communicate is that they might speak different languages, or, in the current jargon, "use different ontologies."

[4]Comparing the ISBNs of the two books would go a long way toward deciding if they are the same, but the ISBN system has been in effect for only thirty years, so it won't apply to many rare books.

[5]in fact, fictitious.

6. Solving this problem at the logical level allows us to focus on the problem of supplying *bridging axioms* between the two vocabularies. These must be supplied by people, although automated tools can play a significant role in tracking bugs, version control, and such.

One might make the objection that real life cannot be so tidily reduced to predicate calculus. Real ontologies contain many hidden presuppositions, which are lost when they are boiled down to dry axioms. This is a serious objection, but we hope it is less likely to hold true in a domain like ours, where everything has got to be pretty formal for agent communication to be possible at all.

This is obviously work in progress. We are in the process of adapting our Unpop planner [7] to handle hierarchical and contingency planning. We are beginning work on an implementation of the ontology merger.

About the Author

Dr. Drew McDermott is a Professor of Computer Science at Yale University. He received a Ph.D. in 1976 from Massachusetts Institute of Technology. He is the coauthor of two textbooks in artificial intelligence, and has a new book coming out from MIT Press on machine consciousness. He is on the editorial board of Artificial Intelligence, and is a Fellow of the American Association for Artificial Intelligence. His research is in robot navigation, planning, and interagent communication.

Contact Information

Dr. Drew McDermott
Yale University
Dept. of Computer Science
51 Prospect Street
New Haven Ct. 06511

203-432-1284
drew.mcdermott@yale.edu
http://www.cs.yale.edu/homes/dvm/

References

[1] D. Box, D. Ehnebuske, G. Kakivaya, A. Layman, N. Mendelsohn, H. F. Nielsen, S. Thatte, and D. Winer. Simple Object Access Protocol (Soap) 1.1. Technical Report, W3C, 2000. Available at http://www.w3.org/TR/SOAP.

[2] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web Services Definition Language (WSDL) 1.1. Technical report, W3C, 2001. Available at http://www.w3c.org/TR/wsdl.

[3] G. Frank, A. Farquhar, and R. Fikes. Building a Large Knowledge Base from a Structured Source. *IEEE Intelligent Systems,* 14(1), 1999.

[4] E. R. Harold and W. S. Means. XML in a Nutshell: A Desktop Quick Reference. O'Reilly & Associates, 2001.

[5] O. Lassila and R. R. Swick. Resource Description Framework (RDF) Model and Syntax Specification. Technical Report, W3C, 1999. Available at http: //www.w3c.org/TR/REC-rdf -syntax.

[6] D. McDermott. The Planning Domain Definition Language Manual. Technical Report 1165, Yale Computer Science, 1998. (CVC Report 98-003).

[7] D. McDermott. Using Regression-match Graphs to Control Search in Planning. *Artificial Intelligence,* 109(1-2):111-159, 1999.

[8] D. McDermott, M. Burstein, and D. Smith. Overcoming Ontology Mismatches in Transactions with Self-Describing Agents. In *Proceedings of Semantic Web Working Symposium,* pages 285-302, 2001.

[9] P. Mitra, G. Wiederhold, and M. Kersten. A Graph-Oriented Model for Articulation of Ontology Interdependencies. In *Proceedings of Conference on Extending Database Technology (EDBT 2000),* 2000.

Army Intelligent Agents for Software Engineering

by C. Ronald Green, U.S. Army Space and Strategic Defense Command

Introduction

Computer technologies are the force multiplier for the Army After Next warfighter. The U.S. Army Space and Strategic Defense Command is developing advanced computer hardware and software technologies to provide the 21st century warfighter with information superiority. Much of this work is being done by the Missile Defense and Space Technology Center's Advanced Technology Directorate in Huntsville, Alabama.

The use of state-of-the-art advanced interface effectors with total spectrum information presentation, autonomous intelligent agents, and self-learning decision aids combined with distributed heterogeneous high-speed processors provide the capability to collect, process, and disseminate a fault tolerant flow of information and knowledge while exploiting or denying an adversary's ability to do the same.

Available Advanced Intelligent Agents

Advanced Intelligent Agents currently available as a result of this research include the: Decision Support Tool, Reuse Decision Maker's Toolset, and Verification, Validation and Accreditation Computer-Based Training Courseware.

Decision Support Tool

The Decision Support Tool (DST) is a PC-based support tool that enables users to improve decision making, record decision rationale, integrate mechanisms, and capture historical data. The current version of the DST has been applied to the mission application segmentation processes of the Defense Information Infrastructure (DII) Common Operating Environment (COE). These processes aid in identifying and developing COE components, and developing COE-based applications. The DST provides the user with a complete description of activities associated with developing a COE software segment through the use of automated tools which reduce user generated input errors. The DST also provides insight into the DII compliance requirements by allowing the user to perform a pre-compliance application test that helps to determine potential cost and schedule impacts.

Reuse Decision-Maker's Toolset

The Reuse Decision-Maker's Toolset (RDT) is a PC-based software tool that automates the Software Reuse Business Model. The RDT provides guidance to program planners and decision makers on performing reuse-based software acquisition. The tool employs a graphical user interface, relational database, and extensive support and help features. The RDT provides program and domain management personnel a detailed process to develop and implement reuse activities throughout the system acquisition life cycle. The tool addresses three perspectives which identify key activities that must be performed and provides mechanisms allowing the capturing of key information regarding decisions made throughout the acquisition process:

- User Perspective - Activities include identifying mission need and assessing reuse activities.
- Domain Perspective - Activities include developing domain infrastructure, developing high-level acquisition strategy, and implementing and maintaining the domain.
- Program Perspective - Activities include identifying requirements, developing, and refining business plan, performing contracting, managing system development, and deploying and maintaining system.

Verification, Validation, and Accreditation (VV&A) Computer-Based Training (CBT) Courseware

The Verification, Validation & Accreditation Computer-Based Training courseware provides broad spectrum VV&A training for the modeling and simulation community with special emphasis on Distributed Interactive Simulations. The courseware is intended to promote uniform application of VV&A

processes Army and DoD-wide. The VV&A CBT course materials are layered into four levels to meet a broad cross-section of skills, backgrounds, and interests:

- Level 1 includes the introduction, main menus, and instructions on how to use the training package.
- Level 2 contains tutorials on VV&A methodology and application, and is targeted at managers, decision makers, planners, and practitioners. Specifically, this level contains a VV&A overview and tutorials on VV&A of DIS, VV&A application tailoring, and requirements for Subject Mater Experts (SME).
- Level 3 provides "how to" instructions for VV&A practitioners. It may also be used by managers or planners who need additional material on a specific subject. It includes detailed training modules which address planning, tailoring, and costing VV&A, personnel issues, documentation requirements, configuration management, and acceptability measures.
- Level 4 provides references to assist users in finding and accessing additional material related to M&S, DIS, and VV&A.

About the Author

Dr. C. Ronald Green is the chief of the computer technologies division within the U.S. Army Space and Missile Defense Command in Huntsville. During his career, Green served as a design engineer with Boeing and Sperry Rand where he worked on control systems for the Saturn V and the B-52.

Dr. Green earned his bachelor's degree in electrical engineering from University of Alabama and both a master's degree in administrative science and a doctorate in computer science from the University of Alabama at Huntsville. Dr. Green is the point of contact for software issues and for transitioning emerging computer technologies within the U.S. Army Space and Strategic Defense Command.

Contact Information

Dr. C. Ronald Green
U.S. Army Space and Strategic Defense Command
CSSD-TC-AS
PO Box 1500
Huntsville, AL 35807-380

205-955-3498

To Subscribe to this Publication Contact:

Phone: 800-214-7921
Fax: 315-334-4964
E-mail: news-editor@dacs.dtic.mil
Web: www.dacs.dtic.mil/forms/regform.shtml

STN Vol. 5, No. 1
In This Issue

Advertisement

The *DoD Software Tech News* is now accepting advertisements for future newsletters. In addition to being seen by the thousands of people who subscribe to the *DoD Software Tech News* in paper copy the advertisement will also be placed on the Data & Analysis Center for Software's website (http://iac.dtic.mil/dacs/) exposing your product, organization, or service to hundreds of thousands of additional eyes.

Interested in learning more? For rates, layout information, and requirements contact:

Dave Nicholls
DACS Deputy Director
Data & Analysis Center for Software
P.O. Box 1400
Rome, NY 13442-1400

Phone: 800-214-7921
Fax: 315-334-4964
E-mail: dnicholls@dacs.dtic.mil

Article Reproduction

Images and information presented in these articles may be reproduced as long as the following message is noted:

> "This article was originally printed in the *DoD Software Tech News*, Vol. 5, No. 1. Requests for copies of the referenced newsletter may be submitted to the following address:
>
> Lon R. Dean, Editor
> Data & Analysis Center for Software
> P.O. Box 1400
> Rome, NY 13442-1400
>
> Phone: 800-214-7921
> Fax: 315-334-4964
> E-mail: news-editor@dacs.dtic.mil
>
> An archive of past newsletters is available at www.dacs.dtic.mil/awareness/newsletters/."

In addition to this print message, we ask that you send us three copies of any document that references any article appearing in the *DoD Software Tech News*.

Data & Analysis Center for Software
http://iac.dtic.mil/dacs

Data & Analysis Center for Software
P.O. Box 1400
Rome, NY 13442-1400

Return Service Requested

PRSRT STD
U.S. Postage
PAID
Permit #566
UTICA, NY